Lab Assignment 1

This assignment includes two parts, Part A is to implement a simple matrix multiplication and Part B is for data parallel reduction. This assignment will be due at 23:59pm, Feb. 7th EST. Please submit your codes and results on Blackboard. The following description is based on Linux platform. If you will do the lab on Windows, you need to change the directories properly.

Part A: Simple Matrix Multiplication (50%)

Download the code simple-matrixmul.zip from the blackboard.

1) Unzip simple-matrixmul.zip into your home directory, please make sure that PATH and LD_LIBRARY_PATH are set correctly.

2) Edit the MatrixMulOnDevice(...) function in matrixmul.cu and the MatrixMulKernel(...) function in matrixmul_kernel.cu to complete the functionality of the matrix multiplication on the device. Do not change the source code elsewhere. The size of the matrix is defined such that one thread block will be sufficient to compute the entire solution matrix.

3) There are several modes of operation for the application.

No arguments: The application will create two randomly initialized matrices to multiply. After the device multiplication is invoked, it will compute the correct solution matrix using the CPU, and compare that solution with the device-computed solution. If it matches (within a certain tolerance), if will print out "Test PASSED" to the screen before exiting.

One argument: The application will use the random initialization to create the input matrices, and write the device-computed output to the file specified by the argument.

Two arguments: The application will initialize the two input matrices with the values found in the files provided as arguments. No output is written to file.

Three arguments: The application will read its inputs from the files provided by the first two arguments, and write its output to the file provided in the third.

Note that if you wish to use the output of one run of the application as an input, you must delete the first line in the output file, which displays the accuracy of the values within the file. The value is not relevant for this application.

4) Answer the following questions:

1. How many times is each element of the input matrices loaded during the execution of the kernel?
2. What is the memory-access to floating-point computation ratio in each thread?

Consider a multiply and addition as separate operations, and ignore the storing of the result. Only global memory loads should be counted towards your off-chip bandwidth

Part B: Data Parallel Reduction (50%)
Download the code from reduction.zip from the blackboard.
1) Unzip reduction.zip into your home directory, please make sure that PATH and LD_LIBRARY_PATH are set correctly.

2) Edit the source files vector_reduction.cu and vector_reduction_kernel.cu to complete the functionality of the parallel addition reduction on the device. The size of the array is guaranteed to be equal to 512 elements for this assignment.

3) There are two modes of operation for the application.
No arguments: The application will create a randomly initialized array to process. After the device kernel is invoked, it will compute the correct solution value using the CPU, and compare that solution with the device-computed solution. If it matches (within a certain tolerance), if will print out "Test PASSED" to the screen before exiting.

One argument: The application will initialize the input array with the values found in the file provided as an argument. In either case, the program will print out the final result of the CPU and GPU computations, and whether or not the comparison passed.

4) Answer the following questions:
1. How many times does your thread block synchronize to reduce the array of 512 elements to a single value?
2. Over the life of the program, how many warps will be adversely affected by SIMD divergence? Remember that SIMD divergence happens when threads within a warp take different code paths through the program.

Grading:
Your submission will be graded on the following parameters.

- Demo/knowledge: Produces correct result output for test inputs
- Functionality: Correct usage of CUDA library calls and C extensions; Correct usage of thread id's.
- Report: Answer all questions.